Języki i metody programowania – Java
INF302W
Wykład 1 (część 2)

Autor

Dr inż. Zofia Kruczkiewicz

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html

Struktura wykładu

1. **Pisanie programu (definiowanie jedynie funkcji main) z użyciem operatorów relacyjnych i logicznych, instrukcje if-else, switch i break, pobieranie danych z listy argumentów, klasa String. Pisanie programu (definiowanie jedynie funkcji main) z wykorzystaniem instrukcji pętli for, while, do-while, continue i break, tablice zawierające nieobiektowe elementy (EL).**
2. **Identyfikacja danych reprezentowanych przez klasy podczas opracowania koncepcji prostego programu obiektowego. Tworzenie programów z użyciem jednej i wielu klas: budowa klasy, konstruktory, metody, zastosowanie składowych statycznych i niestatycznych, operator new, odwołanie do obiektów-operator kropka, wywołanie metod, przeciążenie metod (cd).**

1) Typy danych

Typy całkowite

Typ	Rozmiar	Zakres przechowywanych danych	Wartości domyślne
byte	8 bitów	-128 do 127	0
short	16 bitów	-32768 do 32767	0
int	32 bity	-2147483648 do 2147483647	0
long	64 bity	-9223372036854775808 do 9223372036854775807	0L

Typy rzeczywiste

Typ	Rozmiar	Zakres przechowywanych danych	Wartości domyślne
float	32 bity	1.4E-45 do 3.4E+38	0.0f
double	64 bity	4.9E-324 do 1.7E+308	0.0d

Typ znakowy char

Typ	Rozmiar	Zakres przechowywanych danych	Wartość domyślna
char	16 bitów	Unicode 0 do Unicode 2^{16}-1	'\u0000'

gdzie Unicode służy do kodowania znaków międzynarodowych za pomocą 16 bitów

Typ logiczny boolean

Brak precyzyjnej informacji o rozmiarze; Wartości: false, true; Wartość domyślna: false

2) Zmienne

- **Zmienne typów podstawowych**

np. **int** a;

- **Zmienne typu klasa**

np.

String nazwisko = ”Kowal”; //zmienna nazwisko typu referencja do obiektu zawierającego nazwę Kowal

Znaki łańcucha mogą być kodowane za pomocą kodu UTF-8, kodującego znaki za pomocą 8 bitów (1 bajt), jeśli są to znaki ASCII lub może użyć więcej bajtów, gdy znaki łańcucha nie są kodami ASCII.

Punkt p //referencja do typu Punkt, może być w przyszłości użyta jako odwołanie do obiektu typu Punkt

p = **new** Punkt(); //p jest teraz odwołaniem do obiektu typu Punkt

- **Zmiene ustalone**

final int Init = 1; //nie można zmienić wartości zmiennej ustalonej Init

Punkt = **new** Punkt(Init, Init); //zastosowana do zainicjowania obiektu może poprawić czytelność programu

---

3) Komentarze

- // wyłączenie z programu tekstu od znaku komentarza do końca linii
- /* */ wyłączenie z programu tekstu zawartego między znakami komentarza
- /** */ tworzenie dokumentacji z tekstu zawartego miedzy znakami zawartymi między znakami komentarza za pomocą programu **javadoc**

4) Stałe czyli literały

- **Stałe całkowite** – są traktowane ja stałe typu **int**

Typ	Zmienna	Wartość dziesiętna	Wartość ósemkowa	Wartość szesnastkowa
int	Numer1	320	0500	0x140
long	Numer2	320L lub 320l	0500L	0x140L

- **Stale rzeczywiste** – są traktowane domyślnie jako stałe **typu double**

Typ	Zmienna	Zapis ułamkowy	Zapis wykładnikowy
float	Numer3	2.14F lub 2.14f	21.4e-1F lub 21.4e-1f
double	Numer4	2.24	224e-2

- **Stałe logiczne** typu **boolean** **true i false**
- **Stałe znakowe** typu **char**

Znak	Interpretacja
'\n'	Nowy wiersz
'\t'	Tabulacja pozioma
'\b'	backspace
'\r'	Powrót karetki
'\f'	Wysunięcie papieru
'\\'	Ukośnik lewy
'\''	Znak apostrofu
'\"'	Znak cudzysłowu
'\d'	Liczba w notacji dziesiętnej
'\xd'	Liczba w notacji szesnastkowej
'\ud'	Znak w standardzie Unicode

- **Stałe łańcuchowe** typu String

Są przechowywane jako obiekty typu String

```
String nazwa = ”Zeszyt”;        // obiekt typu String przechowuje znaki ”Zeszyt”
                                //dostęp do obiektu umożliwia zmienna referencyjna nazwa
```

5) Zastosowanie znaków '_' w stałych numerycznych

Wstawia się znaki '_' do stałych numerycznych jedynie w celu poprawy czytelności ich wartości

```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
float pi =  3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
long bytes = 0b11010010_01101001_10010100_10010010;
```

Błędy wstawiania znaków '_':

- Na początku i na końcu liczby
- Przed lub za znakiem kropki w liczbach zmiennoprzecinkowych
- Przez przyrostkiem L lub F
- W miejscach, gdzie oczekiwany jest ciąg cyfr

6) Typy operatorów

Operatory	Notacja
przyrostkowe	*expr*++ *expr*--
jednoargumentowe	++*expr* --*expr* +*expr* -*expr*
jednoargumentowe (negacja logiczna)	!
multiplikatywny (arytmetyczne)	* / %
przylaczeniowe (arytmetyczne)	+ -
relacyjne	< > <= >=
równości	== !=
porównanie typów obiektów	instanceof
jednoargumentowy, negacja bitów	~
bitowy iloczyn logiczny AND	&
bitowa alternatywa logiczna XOR	^
bitowa alternatywa logiczna OR	|
przesuniecie bitowe	<< >> >>>
iloczyn logiczny AND	&&
alternatywa logiczna OR	||
trójkowy (wraunkowe)	? :
przypisanie	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Operatory – ustawione priorytetami

.	wybór składowej	obiekt, składowa
[]	indeksowanie	wskaźnik[wyrażenie]
()	grupowanie wartości	typ(lista_wyrażeń)
++	przyrostkowe zwiększanie o 1	zmienna++
++	przedrostkowe zwiększanie o 1	++zmienna
--	przyrostkowe zmniejszanie o 1	zmienna--
--	przedrostkowe zmniejszanie o 1	--zmienna
!	negacja logiczna	! wyrażenie
-	minus jednoargumentowy	-wyrażenie
instanceof	określanie klasy danego obiektu	obiekt **instaceof** klasa – true lub false
new	utwórz (przydziel pamięć)	**new** typ
*	mnożenie	wyrażenie * wyrażenie
/	dzielenie	wyrażenie / wyrażenie
%	modulo (dzielenie z resztą)	wyrażenie % wyrażenie
+	dodawanie (plus)	wyrażenie + wyrażenie
-	odejmowanie (minus)	wyrażenie - wyrażenie
<<	przesuwanie w lewo	wyrażenie << wyrażenie
>> >>>	przesuwanie w prawo przesuwanie w prawo bez znaku	wyrażenie >> wyrażenie wyrażenie >>> wyrażenie
<	mniejszy	wyrażenie < wyrażenie
<=	mniejszy lub równy	wyrażenie <= wyrażenie
>	większy	wyrażenie > wyrażenie

>=	większy lub równy	wyrażenie >= wyrażenie
==	równy	wyrażenie == wyrażenie
!=	nie równy	wyrażenie != wyrażenie
~	negacja bitowa	-wyrażenie
&	koniunkcja bitowa	wyrażenie & wyrażenie
^	różnica symetryczna	wyrażenie ^ wyrażenie
|	alternatywa bitowa	wyrażenie | wyrażenie
&&	iloczyn logiczny	wyrażenie && wyrażenie
||	suma logiczna	wyrażenie || wyrażenie
? :	wyrażenie warunkowe	wyrażenie ? wyrażenie : wyrażenie
=	proste przypisanie	zmienna = wyrażenie
+=	dodaj i przypisz	zmienna += wyrażenie
-=	odejmij i przypisz	zmienna -= wyrażenie
*=	pomnóż i przypisz	zmienna *= wyrażenie
/=	podziel i przypisz	zmienna /= wyrażenie
%=	weź modulo i przypisz	zmienna %= wyrażenie
^=	różnica bitowa i przypisz	zmienna ^= wyrażenie
&=	koniunkcja bitowa i przypisz	zmienna &= wyrażenie
| =	alternatywa bitowa i przypisz	zmienna | = wyrażenie
<<=	przesuń w lewo i przypisz	zmienna <<= wyrażenie
>>=	przesuń w prawo i przypisz	zmienna >>= wyrażenie
>>>=	przesuń w prawo bez znaku i przypisz	zmienna >>>=wyrażenie

7) Wyrażenia

- **Konkatenacja –** łączenie łańcuchów

Przykłady

```
System.out.print("Dzien dobry, nazywam się Jan Kowalski\n");
System.out.print("Dzien dobry" +  "nazywam się Jan Kowalski\n");
System.out.println("petla "+j);   //j jest traktowana jako łańcuch jednoznakowy
System.out.println("WspolrzednaX = "+ p1.podajX());
```

- **Działania arytmetyczne**

Argumenty o mniejszym rozmiarze typu są przekształcane do typów o większych rozmiarach:
jeden jest typu **double**, drugi jest przekształcany do **double**,
lub jeden jest **float**, drugi jest przekształcany do **float,**
lub jeden jest **long**, drugi jest przekształcany do **long**
lub jeden jest **int**, drugi jest przekształcany do **int**
lub oba są **int**

C/C++			B.Pascal
++	przyrostkowe zwiększanie o 1	zmienna++	inc(x)
++	przedrostkowe zwiększanie o 1	++zmienna	inc(x)
--	przyrostkowe zmniejszanie o 1	zmienna--	dec(x)
--	przedrostkowe zmniejszanie o 1	--zmienna	dec(x)
*	Mnożenie	wyrażenie*wyrażenie	*
/	Dzielenie bez reszty	wyrażenie typu całkowitego /wyrażenie typu całkowitego	div

/	Dzielenie	wyrażenie typu rzeczywistego /wyrażenie typu rzeczywistego	/
%	modulo (dzielenie z resztą)	wyrażenie%wyrażenie	mod
+	dodawanie (plus)	wyrażenie+wyrażenie	+
-	odejmowanie (minus)	wyrażenie-wyrażenie	

Przykłady:

```
public class dzialania                    //klasa publiczna, nieabstrakcyjna, niefinalna
{
  public static void main (String[] args)
 { int  i = 10, j=25, w1;
    double w2;
    w1 = i/j;          System.out.println(w1);      //wartość 0 ( dzielenie bez reszty)
    w1 = j/i;          System.out.println(w1);      //wartość 2 (dzielenie bez reszty)
    w1 = j%i;          System.out.println(w1);      //wartość 5 (reszta z dzielenia)
    w1 = i%j;          System.out.println(w1);      //wartość 10 (reszta z dzielenia)
    w2 =  i/j*1.0;     System.out.println(w2);      //wartość 0.0 ((10/25)*1.0=0*1.0=0.0)
    w2 =  i/(j*1.0);   System.out.println(w2);      //wartość 0.4 (10/25.0=0.4)
   }
}
```

- **Operatory przypisania** (najczęściej używane)

			Znaczenie (przykłady)	
=	proste przypisanie	zmienna = wyrażenie	z=3*y	z=3*y
*=	pomnóż i przypisz	zmienna *= wyrażenie	z*=3*y	z=z*3*y
/=	podziel i przypisz	zmienna /= wyrażenie	z/=3*y	z=z/(3*y)
%=	weź modulo i przypisz	zmienna %= wyrażenie	z%=3*y	z=z%(3*y)
+=	dodaj i przypisz	zmienna += wyrażenie	z+=3*y	z=z+3*y
-=	odejmij i przypisz	zmienna -= wyrażenie	z-=3*y	z=z-3*y

- **Operatory relacyjne** dwuargumentowe

C/C++			B.Pascal
<	mniejszy	wyrażenie < wyrażenie	<
<=	mniejszy lub równy	wyrażenie <= wyrażenie	<=
>	większy	wyrażenie > wyrażenie	<=
>=	większy lub równy	wyrażenie >= wyrażenie	>=
==	równy	wyrażenie == wyrażenie	=
!=	nie równy	wyrażenie != wyrażenie	<>

}

- **Operatory jednoargumentowe**

C/C++			B.Pascal
-	minus jednoargumentowy	-wyrażenie	-
+	plus jednoargumentowy	+wyrażenie	+

- **Operatory logiczne** (rachunek zdań) dwuargumentowe

C/C++			B.Pascal
!	negacja logiczna	! wyrażenie	not
&&	iloczyn logiczny	wyrażenie && wyrażenie	and
||	suma logiczna	wyrażenie || wyrażenie	or
? :	wyrażenie warunkowe	wyr1 ? wyr2 : wyr3 gdzie wyr1 jest typem logicznym, wyr2 i wyr3 są dowolnymi, takimi samymi typami różnymi od **void** np. wynik = x!=0 ? y/x : 0 – wynik może mieć wartość y/x, gdy x!=0 lub wartość 0, gdy x==0	-

8) Instrukcje wyboru if, if else

```
if ( wyrażenie logiczne )  instrukcja;

if ( wyrażenie logiczne )  instrukcja1;
 else  instrukcja2;
```

```
public class  Instrukcja1        //klasa publiczna, nieabstrakcyjna, niefinalna
 {
  public static void main (String[] args)
 {
      int wzrost=172;
      if ( wzrost < 180 )
         if ( wzrost > 175 )
             System.out.println("Wysoki!\n");
         else
             System.out.println("Może być niski!\n");
 }
}
```

```
run:
Może być niski!

BUILD SUCCESSFUL (total time: 0 seconds)
```

```
public class Instrukcja2                    //klasa publiczna, nieabstrakcyjna, niefinalna
 {
  public static void main (String[] args)
 {
    int wzrost=172;
     if ( wzrost < 180 )
        {
          if ( wzrost > 175 )
            System.out.println("Wysoki!\n");
         }
      else
           System.out.println("Jest z pewnością wysoki!\n");
 }
}
```

```
run:
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
public class Instrukcja3                    //klasa publiczna, nieabstrakcyjna, niefinalna
 {
  public static void main (String[] args)
 {
     int wzrost=172;
     if ( wzrost < 180 )
        if ( wzrost > 175 )
            System.out.println("Wysoki!\n");
        else    System.out.println("Może być niski!\n");
     else  System.out.println("Jest z pewnoscia wysoki!\n");
 }}
```

```
run:
Może być niski!

BUILD SUCCESSFUL (total time: 0 seconds)
```

9) Instrukcja wyboru switch

switch (wyrażenie) instrukcja
case stała wyrażenia : **break**
default :

Instrukcja **switch** działa szybciej niż **if else**
Typy danych wyrażenia:

- typy elementarne (byte, short, char, int),
- String, Character, Byte, Short, and Integer

Przykład 1

```
switch (operator) {
  case '*':   x *= y; break;       // instrukcja break powinna zawsze wystąpić
  case '/':   x /= y;  break;      // gdy realizuje się alternatywę
  case '+':   x += y; break;
  case '-':   x -= y;  break;
  case 'p':
  case 't':   x++;    break;
  case 'e':
  case 'r':
  case 'm':   System.out.println ("Tych dzialan kalkulator nie wykona"); break;
  default:    System.out.println ("Pomyłka!");                           break;
 }
```

Przykład 2

```
public class Instrukcja4       //klasa publiczna, nieabstrakcyjna, niefinalna
{
  public static void main (String[] args)
  { double a=1.0, b=2.0;
   char op='/';
    switch ( op )
      {
        case '+':
         case 'd' :      System.out.println(a + b);                          break;
         case '-':       System.out.println(a - b);                          break;
         case '*':       System.out.println(a * b);                          break;
         case '/':       if (b!=0)
                              System.out.println(a / b);
                         else
                              System.out.println(" Dzielenie przez zero");   break;
         default:     System.out.println(" Zly operator");                   break;
     } // koniec switch
    }
  }
```

```
run
0.5
BUILD SUCCESSFUL (total time: 0
```

10) Instrukcje pętli

10.1) while składnia: **while** (wyrażenie logiczne) instrukcja

- W pętli **while** <instrukcja> jest powtarzana tak długo, jak *wyrażenie logiczne* ma wartość równą **true**.
- Test *wyrażenia logicznego* jest zawsze wykonywany przed wykonaniem *instrukcji*.

Przykład :

```
int ile=10;
while (ile >1 ) ile--;                    //pętla wykona się 10
```

10.2) do ... while składnia : **do** instrukcja **while** (wyrażenie logiczne);

- W pętli **do...while** instrukcja jest powtarzana tak długo, jak wartość *wyrażenia logicznego* jest równa **true**.
- Test *wyrażenia logicznego* odbywa się po wykonaniu *instrukcji.*

Przykład

```
public class Instrukcja5 {
 public static void main(String[] args) {
    int ile = 10;
    do {
       ile--;
       System.out.print(ile+" ");
    } while (ile > 1);                    //pętla wykona się 9
} }
```

```
run:
9 8 7 6 5 4 3 2 1  BUILD SUCCESSFUL (total time: 0 seconds)
```

10.3) for składnia: for ([wyr1] ; [wyr_log] ; [wyr2]) instrukcja

- W pętli **for** *instrukcja* jest powtarzana, aż wyrażenie *wyr_log* osiąga wartość równą **false**.
- Przed pierwszą iteracją jest obliczane wyrażenie *wyr1*. Jest ono zazwyczaj używane do inicjowania zmiennej sterującej pętlą. Wyrażenie *wyr1* może być deklaracją.
- Po każdej iteracji po wykonaniu instrukcji wyrażenie *wyr2* jest obliczane. Jest ono zazwyczaj używane do zmiany wartości zmiennej sterującej pętlą.

Wszystkie wyrażenia : *wyr1, wyr_log, wyr2* są opcjonalne:

```
for ( ;  ; );
```

W tym przypadku wyrażenie *wyr_log* ma wartość true (pętla nieskończona).

Przykład – dwie równoważne pętle **for**

```
for (int ile=10; ile>1;ile--);
```

```
for (int ile=10; ile>1;)
    ile--;
```

Pętla for stosowana do przetwarzania kolekcji (pochodnych Collections) i tablic.

```
public class Instrukcja6 {

   public static void main(String[] args) {
      int[] liczby = {1, 2, 3, 4};
      for (int liczba : liczby) {
         System.out.println("Liczba: " + liczba);
      }
   }
}
```

```
run:
Liczba: 1
Liczba: 2
Liczba: 3
Liczba: 4
BUILD SUCCESSFUL (total time: 0 seconds)
```

10.4) Przykłady zastosowania pętli: do while

//(1+1/2+1/3+1/4+1/5.....)

Algorytm:

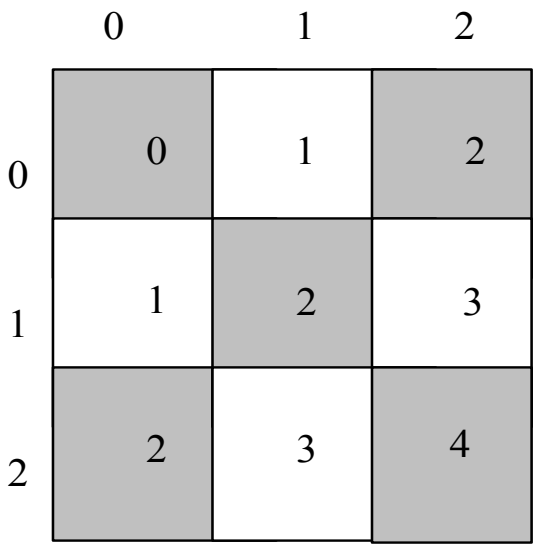
1. Podaj dokładność eps spełniającą warunki: 00.0000001<eps<1, Dokładność oznacza, że suma szeregu podana przez program różni się od poprzedniej o wartość mniejszą niż *eps* (każda suma różni się, ponieważ jest sumą różnej liczby elementów)
2. Zainicjuj sumę szeregu suma =0;
3. Oblicz pierwszy element el=1
4. Zainicjuj licznik elementów szeregu i=2
5. Oblicz sumę szeregu suma = suma + el
   1) Oblicz kolejny element szeregu el=1/i
   2) Zwiększ licznik elementów o 1
   3) Sprawdź, czy kolejny element spełnia warunek el >= eps. Jeśli tak, przejdź do kroku 5, w przeciwnym wypadku przejdź do kroku 6
6. Podaj wartość sumy szeregu

```
//oblicza sumę szeregu harmonicznego

public class szereg                 //klasa publiczna, nieabstrakcyjna, niefinalna
{
public static void main (String[] args)
{ long ii=2;
  double el=1, suma=0, eps=0.0023;
      // double eps=0.0023;
      //double el=1;
      //double  suma=0;
      //long  ii=2;
  do
    {suma=suma+el;
      el=1.0/ii;                    //obliczenia na wartościach double
      ii++;
    } while(el>=eps);               //sumuj tak długo, aż kolejny element sumy będzie mniejszy od
                                    //dokładności, co oznacza, że wartość kolejnej sumy wzrośnie
                                    //poniżej założonej dokładności eps
    System.out.println("Suma elementow szeregu harmonicznego "
                       +suma
                       +" z dokladnoscia " + eps);      //wyświetlanie wartości double
   }
}
```

Suma elementow szeregu harmonicznego 8.954571913334437 z dokladnoscia 2.3E-4

10.5) Pętle zagnieżdżone for, operator ?:

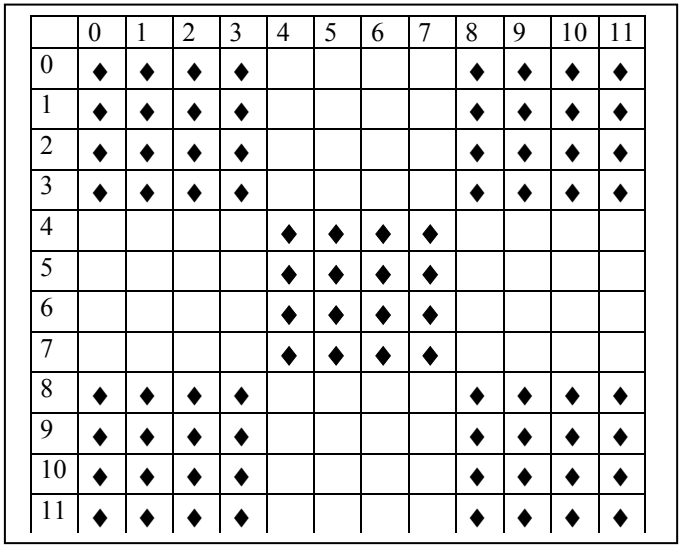


Pierwsza kolumna (x=0)

(0/4+0/4)%2=0
(1/4+0/4)%2=0
(2/4+0/4)%2=0
(3/4+0/4)%2=0

(4/4+0/4)%2=1
(5/4+0/4)%2=1
(6/4+0/4)%2=1
(7/4+0/4)%2=1

(8/4+0/4)%2=0
(9/4+0/4)%2=0
(10/4+0/4)%2=0
(11/4+0/4)%2=0



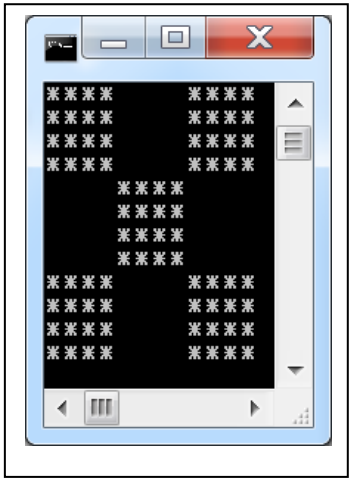
(8/4+0/4)%2=0 (8/4+4/4)%2=1 (8/4+8/4)%2=0
(8/4+1/4)%2=0 (8/4+5/4)%2=1 (8/4+9/4)%2=0
(8/4+2/4)%2=0 (8/4+6/4)%2=1 (8/4+10/4)%2=0
(8/4+3/4)%2=0 (84+7/4)%2=1 (8/4+11/4)%2=0

9-my wiersz (y=8)

```
//klasa publiczna, nieabstrakcyjna, niefinalna
public class Szachownica1 {
   public static void main (String[] args)
  {
   int x, y, k=4;
   for (y=0; y<12; y++)
   { for (x=0; x<12; x++)
        if ((y/k+x/k)%2==0)
            System.out.print("*");
        else
            System.out.print(" ");
     System.out.println( );   }
  }}
```

```
//k: rozmiar pola szachownicy
// rysowanie kolejnej linii szachownicy
// rysowanie kolejnego rzędu w linii szachownicy
//czy suma rząd+kolumna szachownicy parzysta
// jeśli tak - rysowanie zapełnionych pól

// jeśli nie - rysowanie pustych pól
// nowa linia
```



Instrukcja **if else** może być zapisana za pomocą operatora warunkowego **?:**

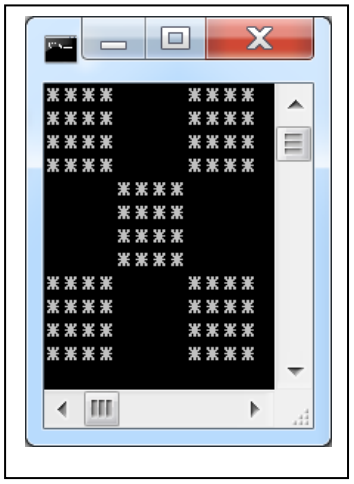
```
if ((y/k+x/k)%2==0)
   System.out.print("*");
else
        System.out.print(" ");
```

```
c=((y/k+x/k)%2)==0 ? '*' : ' ';

System.out.print(c);
```

```
public class Szachownica2 {

   public static void main(String[] args) {
      int x, y, k = 4;                          //k: rozmiar pola szachownicy
      char c;
      for (y = 0; y < 12; y++)                  // rysowanie kolejnej linii szachownicy
      {
         for (x = 0; x < 12; x++)               // rysowanie kolejnej linii rzędu szachownicy
         {
            c = ((y / k + x / k) % 2) == 0 ? '*' : ' ';
            System.out.print(c);
                              //rysowanie zapełnionych pól lub pustych
         }
         System.out.println();                  // nowa linia
      }
   }
}
```

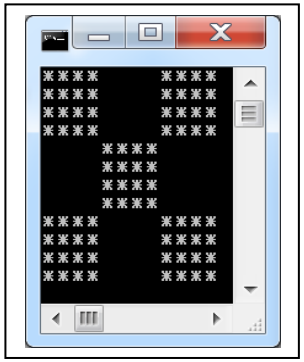


10.6) Instrukcje break i continue dla pętli

```
public class Szachownica3 {

public static void main(String[] args) {
        int x = 0, y = 0, k = 4;                        //k: rozmiar pola szachownicy
        char c;

        for (;;)
        {  if (y == 11)
              break;                                    //po narysowaniu 12 linii koniec rysowania
           x++;
           c = ((y / k + x / k) % 2) == 0 ? '*' : ' ';
           System.out.print(c);                         //narysowanie kolejnego znaku spacji lub "*"
           if (x < 11)                                  //po narysowaniu znaku w kolejnej linii
              continue;          //jeśli jest to 12-y znak, rozpoczęcie kolejnej pętli
           x = 0;                //jeśli nie jest to 12-y znak wyzerowanie liczby znaków w linii
           y++;                  //powiększenie licznika linii o 1
           System.out.println();          //i przejście do następnej linii
         }
     }
  }
```



11. Operatory jednoargumentowe + ,-. Operatory inkrementacji przedrostkowej i przyrostkowej - przykład identyfikacji klas

- Operatory jednoargumentowe – i plus służą do określenia wartości dodatniej lub ujemnej.
- Operatory inkrementacji i dekrementacji przedrostkowej np. ++i lub --i wykonują się najpierw, zanim wykona się wyrażenie, w którym użyto te operatory.
- Operatory inkrementacji i dekrementacji przyrostkowej np. i++ lub i-- wykonują się po wykonaniu wyrażenia, w którym użyto te operatory.

```
import javax.swing.JOptionPane;  //nazwa pakietowa klasy JOptionPane
                                 //czyli nazwa_pakietu1.nazwa_podpakietu1.nazwaklasy
class Operatory_1 {

    int arg11;
    int wynik1;

    public void Przypisz_argument(int arg) {
        arg11 = arg;
    }
```

```
    public String dzialaniajednoarg() {
        String s;
        s = "arg11 : " + arg11+"\n";;
        s += "++arg11 : " + ++arg11 + "\n";       // Pre-increment
        s += "arg11++ : " + arg11++ + "\n";       // Post-increment
        s += "arg11 : " + arg11 + "\n";
        s += "--arg11 : " + --arg11 + "\n";       // Pre-decrement
        s += "arg11-- : " + arg11-- + "\n";       // Post-decrement
        s += "arg11 : " + arg11 + "\n";
        arg11 = -1;
        s += "\narg11 : " + arg11 + "\n";
        s += "++arg11 : " + ++arg11 + "\n";       // Pre-increment
        s += "arg11++ : " + arg11++ + "\n";       // Post-increment
        s += "arg11 : " + arg11 + "\n";
        s += "--arg11 : " + --arg11 + "\n";       // Pre-decrement
        s += "arg11-- : " + arg11-- + "\n";       // Post-decrement
        s += "arg11 : " + arg11 + "\n";
        return s;
    }
}
```

```
class GUI1 { //obsługa wprowadzania i prezentowania danych
    String s;
    public void wyswietlwynik(String s) {
        JOptionPane.showMessageDialog(null, s);
    }
    public int Podaj_daneint() {
        s = JOptionPane.showInputDialog(null, "Podaj argument całkowity");
       return Integer.parseInt(s);
    }
}
public class Operatory1 {      // klasa zarządzająca
    Operatory_1 operatory;
    GUI1 gui;
    public Operatory1(Operatory_1 op, GUI1 gui) {
        operatory = op;
        this.gui = gui;
    }
    public static void main(String[] args) {
     Operatory1 op = new Operatory1(new Operatory_1(), new GUI1()
     op.operatory.Przypisz_argument(op.gui.Podaj_daneint());
     op.gui.wyswietlwynik(op.operatory.dzialaniajednoarg());
    }
}
```